

# SimOutUtils - Utilities for analyzing time series simulation output

Nuno Fachada<sup>1</sup>, Vitor V. Lopes<sup>2</sup>, Rui C. Martins<sup>3</sup>, and Agostinho C. Rosa<sup>1</sup>

<sup>1</sup>*Institute for Systems and Robotics, LARSyS, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal*

<sup>2</sup>*UTEC - Universidad de Ingeniería & Tecnología, Lima, Perú*

<sup>3</sup>*Life and Health Sciences Research Institute, School of Health Sciences, University of Minho, Braga, Portugal*

## Abstract

**SimOutUtils** is a suite of MATLAB/Octave functions for studying and analyzing time series-like output from stochastic simulation models. More specifically, **SimOutUtils** allows modelers to study and visualize simulation output dynamics, perform distributional analysis of output statistical summaries, as well as compare these summaries in order to assert the statistical equivalence of two or more model implementations. Additionally, the provided functions are able to produce publication quality figures and tables showcasing results from the specified simulation output studies.

**Keywords**— Simulation; Modeling; Docking; Simulation output analysis

## (1) Overview

### Introduction

**SimOutUtils** is a suite of MATLAB [1] functions for studying and analyzing time series-like output from stochastic simulation models, as well as for producing associated publication quality figures and tables. More specifically, the functions bundled with **SimOutUtils** allow to:

1. Study and visualize simulation output dynamics, namely the range of values per iteration and the existence or otherwise of transient and steady-state stages.
2. Perform distributional analysis of focal measures (FMs), i.e. of statistical summaries taken from model outputs (e.g., maximum, minimum, steady-state averages).

3. Determine the alignment of two or more model implementations by statistically comparing FMs. In other words, aid in the process of *docking* simulation models [2].
4. From the previous points, produce publication quality L<sup>A</sup>T<sub>E</sub>X tables and figures (the latter via the `matlab2tikz` script [3]).

These utilities were originally developed to study the Predator-Prey for High-Performance Computing (PPHPC) agent-based model [4], namely by statistically analyzing its outputs for a number of different parameters and comparing the dynamical behavior of different implementations [4, 5, 6]. They were later generalized to be usable with any stochastic simulation model with time series-like outputs. The utilities were carefully coded in order to be compatible with GNU Octave [7].

## Implementation and architecture

The `SimOutUtils` suite is implemented in a procedural programming style, and is bundled with a number of functions organized in modules or function groups. As shown in Figure 1, the following function groups are provided with `SimOutUtils`:

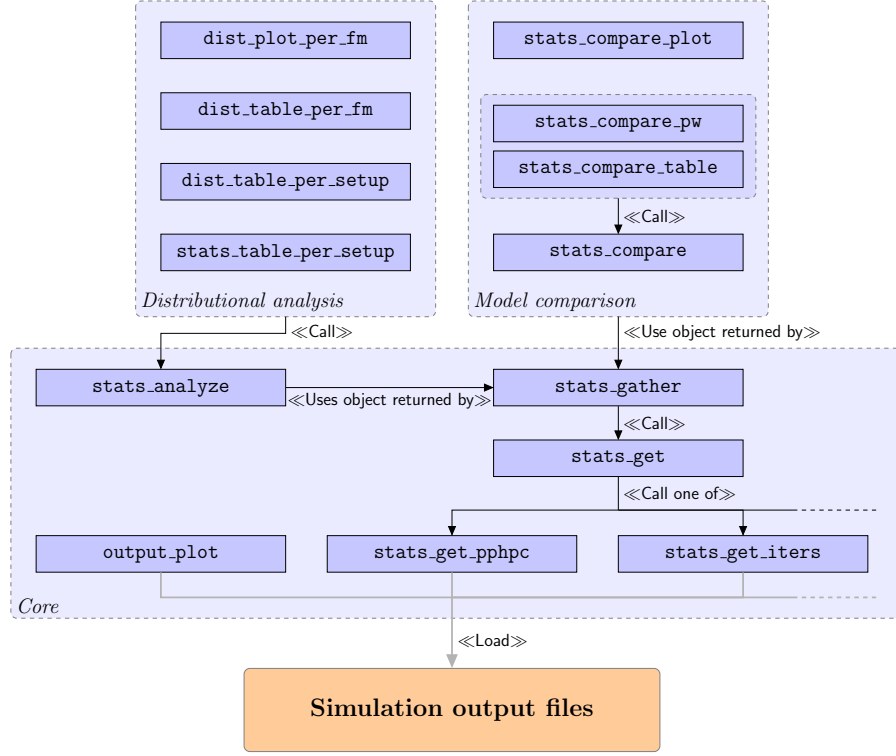
1. Core functions.
2. Distributional analysis functions.
3. Model comparison functions.
4. Helper and third-party functions (not shown in Figure 1).

The next sections describe each group of functions in additional detail.

### Core functions

Core functions work directly with simulation output files or perform low-level manipulation of outputs. The `stats_get` function is the basic unit of this module, and is at the center of the `SimOutUtils` suite. From the perspective of the remaining functions, `stats_get` is responsible for extracting statistical summaries from simulation outputs from one file (i.e., from the outputs of one simulation run). In practice, the actual work is performed by another function, generically designated as `stats_get_*`, to which `stats_get` serves as a facade for. The exact function to use (and consequently, the concrete statistical summaries to extract) is specified in a namespaced global variable defined in the `SimOutUtils` startup script. This allows researchers to extract statistical summaries and use FMs adequate for different types of simulation output.

Two `stats_get_*` functions are provided, namely `stats_get_pphpc` and `stats_get_iters`. The former, set by default, was developed for the PPHPC



**Figure 1** – **SimOutUtils** architecture. Larger blocks with rounded corners and dashed outline constitute function groups, identified in italic font at the lower left corner of the respective block. Within these, functions are represented by smaller blocks with solid outline and sharp corners, with the function name shown in typewriter font. Arrows reflect the relationship between functions and between functions and function groups.

model, and obtains six statistical summaries from each output: maximum, iteration where maximum occurs, minimum, iteration where minimum occurs, steady-state mean and steady-state standard deviation. It is adequate for time-series outputs with a transient stage and a steady-state stage. The latter, **stats\_get\_iters**, obtains statistical summaries corresponding to output values at user-specified instants. It is very generic, and is appropriate for cases where it is hard to derive other meaningful statistics from simulation output. **stats\_get\_\*** functions are also required to provide the name of the returned statistical summaries. This metadata is used by higher level functions for producing figures and tables.

The **stats\_gather** function extracts FMs from multiple simulation output files, i.e., for a number of simulation runs, by calling **stats\_get** for individual files. It returns an object containing a  $n \times m$  matrix, with  $n$  observations (from

$n$  files) and  $m$  FMs (i.e., statistical summaries from one or more outputs). The returned object also includes metadata, namely a data name tag, output names and statistical summary names (via `stats_get` and the underlying `stats_get_*` implementation).

The matrix returned by `stats_gather` can be feed into the `stats_analyze` function, which determines, for each sample of  $n$  elements of individual FMs, the following statistics: mean, variance, confidence intervals,  $p$ -value of the Shapiro-Wilk normality test [8] and sample skewness. This function is called by all functions in the distributional analysis module, as discussed in the next section.

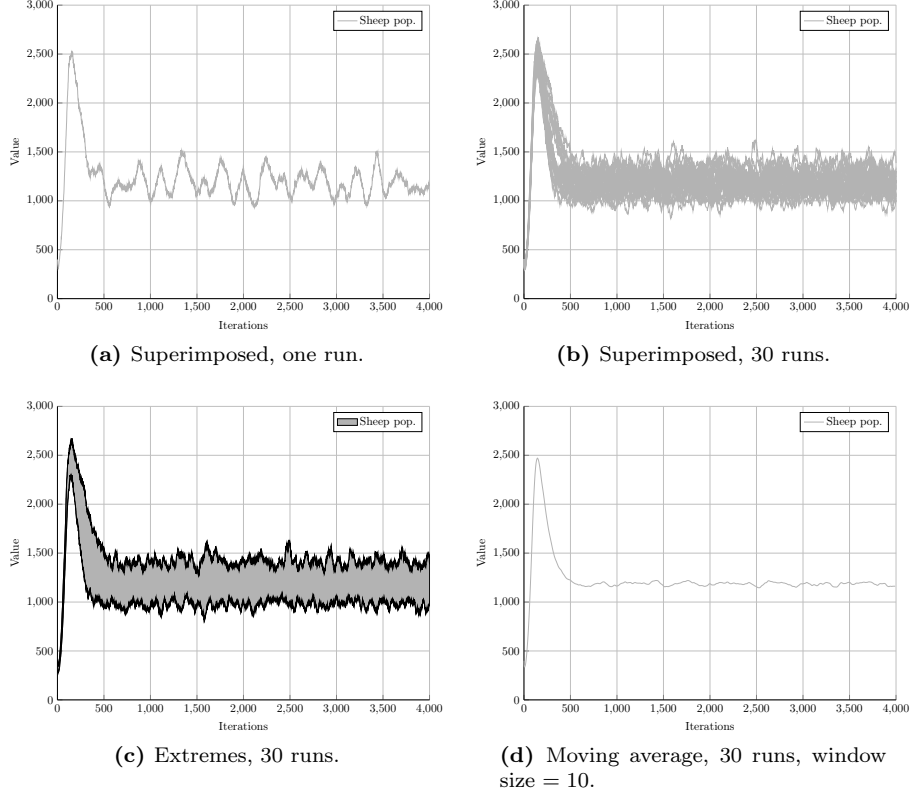
Plots of simulation output from one or more replications can be produced using `output_plot`. This function generates three types of plot: superimposed, extremes or moving average, as shown in Figure 2. Superimposed plots display the output from one or more simulation runs (Figures 2a and 2b, respectively). Extremes plots display the interval of values an output can take over a number of runs for all iterations (Figure 2c). Finally, it is also possible to visualize the moving average of an output over multiple replications (Figure 2d). This type of plot requires the user to specify the window size (a non-negative integer) with which to smooth the output. A value of zero is equivalent to no smoothing, i.e., the function will simply plot the averaged outputs. Moving average plots are useful for empirically selecting a steady-state truncation point.

The provided `stats_get_*` functions, as well as `output_plot`, use the `dlmread` MATLAB/Octave function to open files containing simulation output. As such, these functions expect text files with numeric values delimited by a separator (automatically inferred by `dlmread`). The files should contain data values in tabular format, with one column per output and one row per iteration.

## Distributional analysis functions

Functions in the distributional analysis module generate tables and figures which summarize different aspects of the statistical distributions of FMs. The `dist_plot_per_fm` and `dist_table_per_fm` functions focus on one FM and provide a distributional analysis over several setups or configurations, i.e., over a number of model scales and/or parameter sets. On the other hand, `stats_table_per_setup` and `dist_table_per_setup` offer a distributional analysis of all FMs, fixing on one setup.

The `dist_plot_per_fm` function plots the distributional properties of one FM, namely its estimated probability density function (PDF), histogram and quantile-quantile (QQ) plot. The information provided by `stats_analyze` is shown graphically and textually in the PDF plot. The main goal of `dist_plot_per_fm` is to provide a general overview of how the distributional dynamics of an FM vary with different model configurations. The `dist_table_per_fm` function produces similar content but is oriented towards publication quality materials. It outputs a partial L<sup>A</sup>T<sub>E</sub>X table with a distributional analysis for a range of setups (e.g., model scales) and a specific use case (e.g., parameter set). These partial tables can be merged into larger tables, with custom features such



**Figure 2** – Types of plot provided by the `output_plot` function. All figures show the *sheep population* output from the PPHPC model for size 100, parameter set 1 [4].

as additional rows, headers and/or footers. Tables 8 to 11 of reference [4] were generated with this function.

The `stats_table_per_setup` function produces a plain text or  $\text{\LaTeX}$  table with the statistics returned by the `stats_analyze` function for all FMs for one model setup. In turn, `dist_table_per_setup` generates a  $\text{\LaTeX}$  table with a distributional analysis of all FMs for one model setup. For each FM, the table shows the mean, variance,  $p$ -value of the Shapiro-Wilk test, sample skewness, histogram and QQ-plot. Supplementary Tables S2.1 to S2.10 of reference [4] were created with this function.

### Model comparison functions

Utilities in the model comparison group aid the modeler in comparing and aligning simulation models through informative tables and plots, also producing publication quality  $\text{\LaTeX}$  tables containing  $p$ -values yielded by user-specified

statistical comparison tests.

The `stats_compare_plot` function plots the probability density function (PDF) and cumulative distribution function (CDF) of FMs taken from multiple model implementations. It is useful to visually compare the alignment of these implementations, providing a first indication of the docking process.

The `stats_compare` function is the basic procedure of the model comparison utilities, comparing FMs from two or more model implementations by applying user-specified statistical comparison tests. It is internally called by `stats_compare_pw` and `stats_compare_table`, as shown in Figure 1. The former applies two-sample statistical tests, in pair-wise fashion, to FMs from multiple model implementations, outputting a plain text table of pair-wise failed tests. It is useful when more than two implementations are being compared, detecting which ones may be misaligned. The latter, `stats_compare_table`, is a very versatile function which outputs a  $\text{\LaTeX}$  table with  $p$ -values resulting from statistical tests used to evaluate the alignment of model implementations. It was used to produce Table 8 of reference [5] and Table 1 of reference [6].

## Helper and third-party functions

There are two additional groups of functions, the first containing helper functions, and the second containing third-party functions.

Helper functions are responsible for tasks such as determining confidence intervals, histogram edges, QQ-plot points, moving averages and whether MATLAB or Octave is being used. Functions for formatting real numbers and  $p$ -values, as well as for creating very simple histograms and QQ-plots in  $\text{\textit{TikZ}}$  [9] are also included in this group.

A number of third-party functions, mostly providing plotting features, are also included. The `figtitle` function adds a title to a figure with several subplots [10]. The `fill_between` function [11] is used by `output_plot` for filling the area between output extremes. The `homemade_ecdf` function [12] is a simple Octave-compatible replacement for the MATLAB-specific `ecdf`, assisting `stats_compare_plot` in producing the empirical CDFs. In turn, the `kde` function [13] is used to estimate the PDFs plotted by `stats_compare_plot` and `dist_plot_per_fm`. The `swtest` function is the only third-party procedure not related to plotting, providing the  $p$ -values of the Shapiro-Wilk parametric hypothesis test of normality [14]. Some of these functions were modified, in accordance with the respective licenses, for better integration with the goals of `SimOutUtils`.

## Quality control

All functions have been individually tested for correctness in both MATLAB and Octave, and most are covered by unit tests in order to ensure their correct behavior. The `M0xUnit` framework [15] is required for running the unit tests. Additionally, all the examples available in the user manual (bundled with the software) have been tested in both MATLAB and Octave. These examples

range from simple usage patterns to the concrete use cases of the articles in which `SimOutUtils` was used [4, 5, 6].

## Issues and support

Issues or bugs can be filed at <https://github.com/fakenmc/simoututils/issues>. Support for `SimOutUtils` is provided on best effort basis by emailing the author at [nfachada@laseeb.org](mailto:nfachada@laseeb.org).

## (2) Availability

### Operating system

Any system capable of running MATLAB R2013a or GNU Octave 3.8.1, or higher.

### Programming language

MATLAB R2013a or GNU Octave 3.8.1, or higher.

### Dependencies

MATLAB requires the Statistics Toolbox.

### List of contributors

The software was created by Nuno Fachada.

### Software location

#### Archive

**Name** `SimOutUtils`

**Persistent identifier** <http://dx.doi.org/10.5281/zenodo.50525>

**Licence** MIT License

**Publisher** Zenodo

**Date published** 26/04/2016

## Code repository

**Name** SimOutUtils

**Identifier** <https://github.com/fakenmc/simoututils>

**Licence** MIT License

**Date published** 26/04/2016

## Language

English.

## (3) Reuse potential

These utilities can be used for analyzing any stochastic simulation model with time series-like outputs. As described in ‘Core functions’, output-specific FMs can be defined by implementing a custom `stats_get_*` function and setting its handle in the `simoututils_stats_get_` global variable. The core `stats_gather` and `stats_analyze` functions can be integrated into other higher-level functions to perform operations not available in `SimOutUtils`.

## Acknowledgements

This software uses additional MATLAB/Octave functions written by Chad A. Greene [10], Benjamin Vincent [11], Mathieu Boutin [12], Zdravko Botev [13] and Ahmed Ben Saïda [14].

## Funding statement

This work was supported by the Fundação para a Ciência e a Tecnologia (FCT) projects UID/EEA/50009/2013 and UID/MAT/04561/2013, and partially funded with grant SFRH/BD/48310/2008, also from FCT. The author Vitor V. Lopes acknowledges the financial support from the Prometeo project of SENESCYT (Ecuador).

## References

- [1] The MathWorks, Inc. Natick, Massachusetts, USA MATLAB and Statistics Toolbox Release 2013a (2013).
- [2] Axtell, R., Axelrod, R., Epstein, J. M., and Cohen, M. D. (1996) Aligning simulation models: a case study and results. *Computational and Mathematical Organization Theory*, **1**(2), 123–141.



- [3] Schlömer, N. (2008) matlab2tikz, available: <http://www.mathworks.com/matlabcentral/fileexchange/22022-matlab2tikz-matlab2tikz>.
- [4] Fachada, N., Lopes, V. V., Martins, R. C., and Rosa, A. C. (November, 2015) Towards a standard model for research in agent-based modeling and simulation. *PeerJ Computer Science*, **1**, e36.
- [5] Fachada, N., Lopes, V. V., Martins, R. C., and Rosa, A. C. (January, 2016) Parallelization strategies for spatial agent-based models. *International Journal of Parallel Programming*, pp. 1–33.
- [6] Fachada, N., Lopes, V. V., Martins, R. C., and Rosa, A. C. (March, 2016) Model-independent comparison of simulation output. *arXiv*, **1509.09174** [cs.OH].
- [7] Eaton, J. W., Bateman, D., Hauberg, S., and Wehbring, R. (March, 2015) GNU Octave version 4.0.0 manual: a high-level interactive language for numerical computations, CreateSpace Independent Publishing Platform, fourth edition.
- [8] Shapiro, S. S. and Wilk, M. B. (December, 1965) An analysis of variance test for normality (complete samples). *Biometrika*, **52**(3/4), 591–611.
- [9] Tantau, T. The TikZ and PGF packages Institut für Theoretische Informatik, Universität zu Lübeck (2013).
- [10] Greene, C. A. (2013) Figtitle, available: <http://www.mathworks.com/matlabcentral/fileexchange/42667-figtitle>.
- [11] Vincent, B. (2014) Fill\_between, available: <http://www.mathworks.com/matlabcentral/fileexchange/47151-fill-between>.
- [12] Boutin, M. (2011) Homemade ECDF, available: <http://www.mathworks.com/matlabcentral/fileexchange/32831-homemade-ecdf>.
- [13] Botev, Z. I., Grotowski, J. F., and Kroese (2010) Kernel density estimation via diffusion. *The Annals of Statistics*, **38**(5), 2916–2957.
- [14] Saïda, A. B. (2007) Shapiro-Wilk and Shapiro-Francia normality tests, available: <http://www.mathworks.com/matlabcentral/fileexchange/13964-shapiro-wilk-and-shapiro-francia-normality-tests>.
- [15] Oosterhof, N. N. (2015) MOxUnit - An xUnit framework for Matlab and GNU Octave, available: <http://www.mathworks.com/matlabcentral/fileexchange/54417-moxunit>.